(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号

特開平9－223027

(43)公開日　平成９年(1997)８月26日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/46	３４０		Ｇ０６Ｆ　9/46	３４０Ａ	
				Ｃ	
13/10	３３０		13/10	３３０Ｂ	

審査請求　未請求　請求項の数１　ＯＬ　（全 13 頁）

(21)出願番号　特願平8－340908

(22)出願日　平成８年(1996)12月20日

(31)優先権主張番号　５９３，３１３

(32)優先日　1996年１月31日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　590000400
ヒューレット・パッカード・カンパニー
アメリカ合衆国カリフォルニア州パロアルト　ハノーバー・ストリート　3000

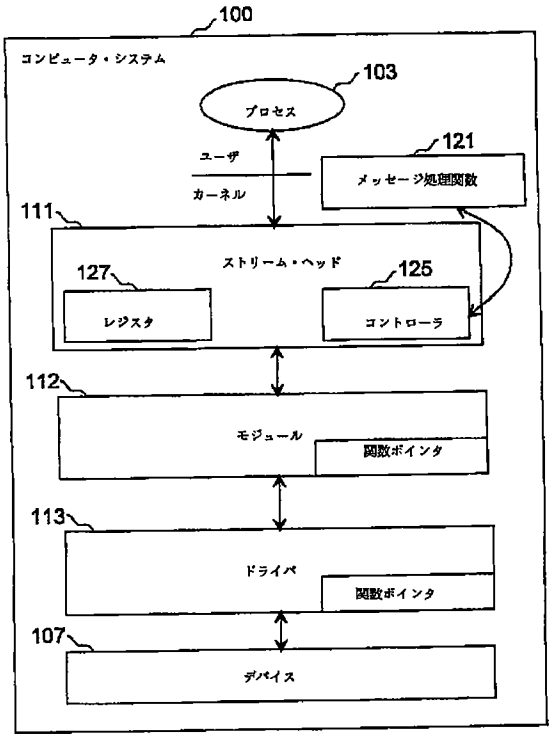
(72)発明者　ヨシヒロ・イシジマ
アメリカ合衆国95120カリフォルニア州サン・ノゼ、クアル・クリーク・サークル　1051

(72)発明者　マイケル・クラウス
アメリカ合衆国95006カリフォルニア州ボールダー・クリーク、ベアー・クリーク・ロード　17523

(74)代理人　弁理士　岡田　次生

(54)【発明の名称】　メッセージ通信装置

(57)【要約】
【課題】システム・アーキテクチャに従って固定的であったＳＴREＡＭＳの機能に、ユーザ・プロセスまたはデバイス・ドライバの動的要求に応答して適切な機能を提供することができる柔軟性を与える。
【解決手段】デバイスとユーザ・プロセスの間でメッセージを通信するためのデータ経路を有するコンピュータ・システムにおいて、デバイスとのメッセージ通信を行えるようにデバイスに接続され、メッセージを処理するメッセージ処理関数を識別するように特に適合されたデバイス・ドライバ、およびユーザ・プロセスとメッセージを通信するためデバイス・ドライバとユーザ・プロセスの間に配置され、上記デバイス・ドライバによって識別された上記メッセージ処理関数の実行を制御する関数コントローラを含むように特に適合されたストリーム・ヘッドを備える装置を提供する。



1

【特許請求の範囲】
【請求項１】デバイスとユーザ・プロセスの間でメッセージを通信するためのデータ経路を有するコンピュータ・システムにおいて、
上記デバイスとメッセージを通信するため該デバイスに接続され、メッセージを処理するメッセージ処理関数を識別するように特に適合されたデバイス・ドライバと、
上記ユーザ・プロセスとメッセージを通信するため上記デバイス・ドライバと上記ユーザ・プロセスの間に接続され、上記デバイス・ドライバによって識別された上記メッセージ処理関数の実行を制御する関数コントローラを含むように特に適合されたストリーム・ヘッドと、
を備える装置。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、データ通信に関するもので、特に、デバイスとホスト・コンピュータの間、同一コンピュータ上で実行する異なるプロセスの間、および異なるコンピュータの上で実行する複数プロセスの間のデータ通信を提供するＳＴＲＥＡＭＳフレーム・ワークに関するものである。
【０００２】
【従来の技術】ＳＴＲＥＡＭＳは、種々のタイプのデバイス・ドライバと同様に、データ通信ネットワーク・プロトコールを実施するための事実上の業界標準フレーム・ワークになっている。ＳＴＲＥＡＭＳは、アプリケーション・プログラムのようなユーザ・レベル・プロセスとカーネル内のデバイス・ドライバの間の双方向性データ経路を提供するストリームである。典的型ストリームは、ストリーム・ヘッド、オプションである処理モジュールおよびデバイス・ドライバという主要な３つのタイプの処理エレメントを含む。
【０００３】デバイス・ドライバは、ストリームの終端すなわち末尾に置かれる。ドライバは、そのオブジェクト・ファイルをカーネル・オブジェクト・ファイルに単にリンクすることによってカーネルに加えられる。しかしながら、従来技術の教示するところによれば、ストリーム・ヘッドは、カーネル固有の特性を備えていているので、固定化されている。
【０００４】ストリーム・ヘッドは、(「ユーザ空間」として知られる)ユーザ・プロセス・レベルとストリームの残りの部分の間のインターフェースを提供し、
(「カーネル空間」として知られる)カーネル・プロセス・レベルにおいてのみ実行する一組のルーチンを含む。従来技術のストリーム・ヘッドの各々は、それがサポートする限られた処理オプションの中から選択できる小さい範囲に対してのみカストマイズすることが可能であるに過ぎない。従来技術においては、ストリーム・ヘッド処理ルーチンは、システムのあらゆるストリームについて使用される。

2

【０００５】ストリームへのアクセスは、ＳＴＲＥＡＭＳファイル記述子を持つopen(オープン)システム呼び出しを介して提供される。例えば、ユーザ・プロセスがシステム呼び出しを行うと、ストリーム・ヘッド・ルーチンが起動され、データのコピー、メッセージ生成および制御動作が実行される。デバイス・ドライバがまだオープンされていなければ、オープン・システム呼び出しがストリームを構築する。一旦ストリームが構築され、オープンが成功裡に完了すれば、同一デバイスに対する別のオープン呼び出しは、同一のストリームを参照する新しいファイル記述子を作成する。
【０００６】ストリーム・ヘッドは、ユーザ・プロセス・レベルとカーネル・プロセス・レベルの間でデータをコピーすることができるストリーム中の唯一のコンポーネントである。その他すべてのコンポーネントは単にメッセージを通信することによってデータ転送を実行するだけなので、それらコンポーネントはユーザ・プロセスとの直接の対話から隔離されている。このように、オープン・ストリームのモジュールおよびドライバは、カーネル・プロセス・レベルの文脈の範囲内でのみ実行するので、ユーザ・プロセス・レベルの文脈または情報の知識を持たない。このような文脈欠如は、ユーザ・プロセスとカーネル・コンポーネントの間の通信を処理するために「１サイズですべてを賄う」アプローチを促進する。そのようなアプローチは、オープンするデバイスに関して現在どのようなユーザ・プロセスが実行していようともそれらプロセスの変更要求を充足するようにモジュールおよびドライバのストリーム・データ解釈、ストリーム実行行動等を適合させる上で制約を課すものである。例えば、従来技術のＳＴＲＥＡＭＳにおけるモジュールおよびドライバは、３２ビット・カーネルおよびプロセッサ・アーキテクチャ上で実行される３２ビット・アプリケーションから、６４ビット・カーネルおよびプロセッサ・アーキテクチャ上で実行される６４ビット・アプリケーションへの移行をサポートする適応面において、特に制約を持つ。６４ビット・カーネルに移行する場合コンピュータ・ユーザが３２ビット・ユーザ・アプリケーションに対して払わなければならない投資を防ぐため、３２ビット・ユーザ・アプリケーションと６４ビット・ユーザ・アプリケーションの両方が６４ビット・カーネル上の実行をサポートされるように、ＳＴＲＥＡＭＳは適合されなければならない。オープン・ストリームのモジュールおよびドライバが上述の例のような６４ビット・カーネルの文脈の範囲内でのみ実行し、従って、３２ビット・アプリケーションの文脈知識を欠如しているので、それらモジュールおよびドライバは、６４ビット・アプリケーションの要求事項と相違する３２ビット・アプリケーションの要求事項を充足するようにストリーム・データ解釈、ストリーム実行行動などを適合することが制約される。

【０００７】一般的に、データに接触する時は必ず、アプリケーションおよびカーネルの処理性能は劣化する。従って、データがストリーム・ヘッドにおいてアクセスされる時は必ず、ストリーム・ヘッドにおいて可能な限り多数の相互に排除的な動作を並列的に実行すべきであるということがすぐれたガイドラインであり、それによって、下流においてデータに再び接触されなければならない必要性が減少する。

【０００８】

【発明が解決しようとする課題】従って、デバイスまたはユーザ・プロセスの動的要求に応じたメッセージ処理の実行をストリーム・ヘッドにおいて制御することができる装置および方法が必要とされる。すなわち、カーネル・レベル・ストリーム処理モジュールまたはデバイス・ドライバに関するメッセージ処理関数を識別し、デバイスまたはユーザ・プロセスの要求のような種々のシステム要求に従ってストリーム・データ解釈、ストリーム実行行動等を適合するためそれら諸関数がストリーム・ヘッドにおいて使用可能となるように、ストリーム・ヘッドにおけるそれら識別された諸関数の実行を制御する装置および方法が必要とされる。

【０００９】

【課題を解決するための手段】本発明は、カーネル・レベル・ストリーム処理モジュールまたはデバイス・ドライバに関するメッセージ処理関数を識別し、デバイスまたはユーザ・プロセスの要求のような種々のシステム要求に従ってストリーム・データ解釈、ストリーム実行行動等を適合するためそれら諸関数がストリーム・ヘッドにおいて使用可能となるように、ストリーム・ヘッドにおけるそれら識別された諸関数の実行を制御する装置および方法を提供する。

【００１０】一般的には、本発明は、デバイス・ドライバまたはストリーム処理モジュールによって識別されるメッセージ処理関数の実行を制御する関数コントローラを含むように特別に適合されたストリーム・ヘッドを含む。好ましくは、デバイス・ドライバまたはストリーム処理モジュールは、例えばデバイス・ドライバがストリームを構築するためオープンされる時、あるいは、ストリーム処理モジュールがストリーム上で使用可能にされる時、ストリームのメッセージを処理する関数をストリーム・ヘッドに初期的に登録する。代替的形態としては、ドライバまたはモジュールは、ストリーム・ヘッドに１つまたは複数のパラメータを渡して、メッセージ毎に関数を起動する。

【００１１】特に利点を持つ局面として、本発明は、データ形式変換関数を識別して、３２ビット・ユーザ・アプリケーションの要求に従ってストリーム・データ解釈、ストリーム実行行動等を適合するようにストリーム・ヘッドにおける関数の実行を制御する６４ビット・カーネル・モジュールまたはドライバを提供する。同様に、本発明は、プロトコール変換関数を識別して、ユーザ・アプリケーションのプロトコール形式要求に従ってメッセージ・プロトコール形式を変換するようにストリーム・ヘッドにおける関数の実行を制御するモジュールまたはドライバを提供する。

【００１２】更に、関数を識別するモジュールまたはドライバによって、また、本発明に従ってストリーム・ヘッドにおける関数の実行を制御することによって、可能な限り多数の相互に排除的な動作が関数によって並列的に実行され、そのため、下流においてデータに接触する必要性が減少する。例えば、ネットワーキング・プロトコールに関して最も共通的な必要タスクの１つは、データ・パケットを送信する際にデータ完全性を維持するためチェックサムを計算することである。チェックサムをサポートしていない従来技術の標準的コピー関数copyin()とは対照的に、本発明は、カーネル・レベル・モジュールまたはドライバ特有の関数を識別し、該関数のストリーム・ヘッドにおける実行を制御して、必要なチェックサムを平行して生成している間にデータをユーザ空間からカーネル空間に移動させる関数が使用可能とさせる。

【００１３】加えて、本発明は、関数を識別するモジュールまたはドライバを提供し、カーネル内のモジュールまたはドライバの状態の動的変化に応答してストリーム・ヘッドにおける関数の実行を制御する手段および方法を提供する。例えば、本発明は、接続ネットワークから受け取る新しい情報に基づくモジュールまたはドライバの優先状態の変更に応答して、ストリーム・ヘッド上で通常の優先度を持つメッセージとして既に待ち行列に入れられたメッセージを優先処理すべきメッセージに変換する関数を識別する手段および方法を提供する。

【００１４】

【発明の実施の形態】本発明の好ましい実施形態のブロック図が図１に示されている。適切なソフトウェアによって制御された１つまたは複数のマイクロプロセッサを好ましくは含むコンピュータ・システム１００の範囲内において、ストリームは、ユーザ・プロセス１０３と例えばハードウェア装置または疑似装置のようなデバイス１０７の間におけるメッセージの双方向通信を提供する。３つの主要タイプのストリーム処理エレメントは、ストリーム・ヘッド１１１、オプションである処理モジュール１１２、および例えばハードウェア・デバイス・ドライバまたは疑似デバイス・ドライバのようなデバイス・ドライバ１１３である。好ましい実施形態において、本発明は、ストリーム・ヘッドにおいてメッセージ処理関数を登録することができるようにＳＴＲＥＡＭＳフレーム・ワークを拡張する。本発明に関連したＳＴＲＥＡＭＳの詳細は、UNIX System V Network Programming by S. Rago,Addison Wesley professional computing series(1993)のChapter 3およびChapter 9ないし11に

記載されている。

【００１５】図１に示されているように、デバイス・ドライバ１１は、デバイスとの間でメッセージを通信するためデバイスに接続されている。モジュールおよびドライバは、メッセージを通信することによってのみデータ転送を実行するので、ユーザ・アプリケーション・プログラムのようなユーザ・プロセスとのいかなる直接対話からも隔離されている。ストリーム・ヘッドは、ユーザ・プロセスとのメッセージ通信を行うためデバイス・ドライバとユーザ・プロセスの間に配置される。ＳＴＲＥＡＭＳフレーム・ワークの範囲内において、上記３つのタイプの処理エレメントの中で、ストリーム・ヘッドが、ユーザ・プロセス・レベルとカーネル・プロセス・レベルの間でデータをコピーすることができる唯一の処理エレメントである。詳細は後述するが、本発明は、１つまたは複数のメッセージ処理関数１２１を識別するカーネル・レベルのモジュールまたはドライバを備える。好ましくは、メッセージ処理関数の各々は、図１に示される少くとも１つの関数ポインタによって識別される。ストリーム・ヘッドは、識別されたメッセージ処理関数のストリーム・ヘッドにおける実行を制御するコントローラ１２５を含む。

【００１６】本発明の好ましい実施形態において、デバイス・ドライバがストリームを構築する時、あるいは、ストリーム処理モジュールがストリーム上で使用可能とされる時、ストリーム・ヘッドにおいてメッセージ処理関数を登録するため、ドライバまたはモジュールは、ストリーム・ヘッドにおけるコントローラに関数ポインタを伝える。従って、好ましい実施形態において、ストリーム・ヘッドは、レジスタ・ブロックまたは１つのレジスタ１２７を含み、そこに関数ポインタを記録することによってメッセージ処理関数の識別情報を記録する。代替的には、ドライバまたはモジュールは、メッセージ毎に１つまたは複数のパラメータをストリーム・ヘッドに渡してメッセージ処理関数を起動させる。従って、デバイス１０７またはユーザ・プロセス１０３の要求事項のようなシステム要求事項に従って、ストリーム・データ解釈、ストリーム実行行動等を適合することができるように関数がストリーム・ヘッドにおいて使用可能とされる。

【００１７】好ましくは、ストリーム・ヘッドのレジスタ・ブロックおよびコントローラは、モジュールまたはドライバによって登録された関数に対応する関数ポインタを記録するパブリック・データ構造を含む適切なソフトウェアを使用して該コンピュータ・システムの範囲内で実現される。パブリック・データ構造は、モジュールまたはドライバによって登録された関数に対応する関数ポインタを保持する。関数ベクトルが、現在の関数登録状態を反映する。従って、ストリーム・ヘッド命令<stream.h>の範囲内に以下のようなパブリック・データ構造が定義される。

【００１８】

【表１】

```
struct sth_func_reg{
int (*sth_copyin)();          /* 書込側copyin */
int sth_copyin_threshold;     /* sth_copyin()に渡されるパラメータ */
int (*sth_32_to_64)();        /* 書込側32-to-64ビット変換関数 */
int (*sth_write_opt)();       /* 書込側オプション関数 */
int (*sth_read_opt)();        /* 読取側オプション関数 */
int (*sth_ioctl_opt)();       /* ioctlオプション関数 */
int (*sth_64_to_32)();        /* 読取側64-to-32ビット変換関数 */
}
```

【００１９】ストリーム・ヘッド宣言(すなわちプライベート構造)の範囲内において、以下の構造が定義される。

【００２０】

【表２】

```
struct sth_s {
...
struct sth_func_reg_sth_f_reg;
...
}
```

【００２１】図２は、本発明の好ましい実施形態の動作の流れを例示している。好ましい実施形態において、ストリーム・モジュールまたはドライバは、str_func_install()を介して関数を登録する。モジュールまたはドライバの導入時、またはモジュールまたはドライバが関数を登録することを決断する時はいつでも、str_func_install()が呼び出される。例えば、ドライバＡが、書込みまたは読取り経路上で使用するアプリケーションのため、３２から６４ビットへおよび６４ビットから３２ビットへの変換関数を識別する。これらの関数はそのドライバがオープンされる時登録される。これによって、単一のドライバ・インスタンスが３２ビットおよび６４ビット・アプリケーションを同時にサポートすることが可能となる。同様に、モジュール・イネーブル(使用可能)動作が発生する場合、ＳＴＲＥＡＭＳは、モジュールのオープン・ルーチンを起動して、ストリーム・ヘッド登録を実行する。

【００２２】図２に示されるように、モジュールが関数

ポインタを使用してsth_32_to_64関数を識別して、記録のためストリーム・ヘッドのコントローラにその関数ポインタを転送する。モジュールがメッセージを送ることによって該関数を使用不可にする場合もある。例えば、モジュールはメッセージを送ることによって関数を使用不可にして、その結果、

sth->sth_func_reg.sth_32_to_64 = NULL

と表現されるようにデータ構造中におけるレコードの評価が無(ヌル)になる。

【００２３】２つのＳＴＲＥＡＭＳコンポーネントが、例えば２つのsth_read_opt()関数のような矛盾するメッセージ処理関数を登録した場合、ストリーム・ヘッド構造は、スタックの範囲内に最高位のコンポーネント関数を反映するであろう。矛盾がなければ、ストリーム・ヘッドは両方のコンポーネントの関数機能を反映するであろう。好ましい呼び出しシーケンスは次の通りである。

【００２４】

【表３】str_func_install(struct sth_func_reg*func_reg,int func_mask);func_maskは、次のいずれかから構成されるビット・マスクである。

【００２５】

【表４】

```
/* func-maskのためのフラグ */
#define FUNC_REG_COPYIN_ENABLE      0X0001
#define FUNC_REG_COPYIN_DISABLE     0X0002
#define FUNC_REG_32_TO_64_ENABLE    0X0004
#define FUNC_REG_32_TO_64_DISABLE   0X0008
#define FUNC_REG_WRITE_OPT_ENABLE   0X0010
#define FUNC_REG_WRITE_OPT_DISABLE  0X0020
#define FUNC_REG_READ_OPT_ENABLE    0X0040
#define FUNC_REG_READ_OPT_DISABLE   0X0080
#define FUNC_REG_64_TO_32_ENABLE    0X0100
#define FUNC_REG_64_TO_32_DISABLE   0X0200
#define FUNC_REG_IOCTL_OPT_ENABLE   0X0400
*#define FUNC_REG_IOCTL_OPT_DISABLE  0X0800
```

【００２６】この呼出しシーケンスは、オープン時またはモジュール起動時にどの関数が自動的に登録されるかを指定する。func_regおよびfunc_maskは、そのモジュール・スイッチ・テーブル・エントリに次のように記憶される。

【００２７】

【表５】

```
struct modsw{
struct modsw*    d_next;
struct modsw*    d_prev;
char             d_namee[FMNAMESZ+1];
char             d_flags;
SQHP             d_sqh;
int              d_curstr;
struct streamtab*d_str;
struct streamtab*d_default_alt;
int              d_naltstr;
struct streamtab**d_altstr;
int              d_sq_level;
int              d_refcnt;
int              d_major;
struct sth_s*    d_pmux_top;
int              d_func_mask;
struct sth_func_reg*d_func_reg;
};
```

【００２８】モジュールまたはドライバは、M_SETOPTSメッセージを通してストリーム・ヘッドに通知することによって登録されたメッセージ処理関数を起動または停止させる。これは、<stream.h>内に定義されたstroptionsデータ構造を拡張することによって行われる。

【００２９】

【表６】

```
struct stroptions {
ulong so_flags;        /* セットすべきオプション */
short so_readopt;      /* 読取オプション */
ushort so_wroff;       /* 書込オフセット */
long so_minpsz;        /* 最小読取パケット・サイズ */
long so_maxpsz;        /* 最大読取パケット・サイズ */
ulong so_hiwat;        /* 読取待ち行列上限マーク */
ulong so_lowat;        /* 読取待ち行列下限マーク */
unsigned char so_band; /* 上下限マークの幅 */
```

【００３０】このデータ構造を次のように拡張すること※によって、関数登録をサポートしないモジュールまたはドライバが従来通り動作し続けるようにオリジナルのレイアウトが維持される。

【００３１】

【表７】

```
struct stroptions {
ulong so_flags;        /* セットすべきオプション */
short so_readopt;      /* 読取オプション */
ushort so_wroff;       /* 書込オフセット */
```

```
longso_minpsz;          /* 最小読取パケット・サイズ */
longso_maxpsz;          /* 最大読取パケット・サイズ */
ulong so_hiwat;         /* 読取待ち行列上限マーク */
ulong so_lowat;         /* 読取待ち行列下限マーク */
unsigned char so_band; /* 上下限マークの幅 */
short so_device_id;     /* ドライバまたはモジュール識別子 */
short so_func_mask;    /* イネーブルされた関数 */
};
```

【００３２】so-device-idは、ドライバまたはモジュールがシステム内に導入される時返されるデバイス識別子である。好ましい実施形態に関して、これはstr_install()経由で実行される。str_install()は、関数登録を実行することを望む時はいつでも利用することができる広域変数にドライバまたはモジュールが記録するユニークなso_device_idを生成する。so_func_maskは、どの関数が起動されるべきかあるいは動作停止されるべきかを指定する。ビットが定義されていない場合は、対応する登録されたメッセージ処理関数は有効のままとされる。

【００３３】モジュールまたはドライバは、それらがストリーム・ヘッド特性を修正するために選択するいかなる時点においても、特別に適合されたM_SETOPTSメッセージを送出する能力を持つ。この構造が解読される時、ストリーム・ヘッドのコードは、so_func_maskを検査して、セットされたフラグに基いて対応するストリーム・ヘッド関数アドレスを記録するかあるいは無効にする。関数が使用可能とされるかあるいは使用可能のままとされる場合、ストリーム・ヘッドは、フラグをセットして、ストリーム・ヘッド・メッセージ処理の間に関数登録が検査されなければならないことを標示する。

【００３４】システム呼び出しと連係して本発明を使用するため、以下のような適合が行われる。write()システム呼び出しに関して、下記のコードがＳＴＲＥＡＭＳ実施形態に追加される。登録される書込み関数は、好ましくは、ユーザ空間からカーネル空間へのデータの移動を自動チェックサム計算と結合するプロトコール依存型＊チェックサム・オフロード関数を含む。この技術を使用することによって、本発明は、ＳＴＲＥＡＭＳフレーム・ワークに付加価値を与えながらプロトコールからの独立を維持し続ける。

【００３５】好ましくは、そのようなコピー／チェックサム(copyin/checksum)結合関数は、処理性能を向上させるためしきい値変数を参照しなければならない。このしきい値変数はオリジナルのデータ構造の一部として記憶され、関数に渡される。次に、関数はuniopの長さフィールドを調べ、しきい値変数と比較して、登録されたcopyin関数を実行することが適切か否かを判断する。上記動作を行わない環境はプロトーコル依存型であるが、基本プロトコールが長さ５１２バイトのような小さいパケットのみを送り出すことを許容し、アプリケーションが８Ｋバイト単位のデータを送り出すような場合が、このようなケースであろう。そのようなケースでは、１６個の登録したコピー／チェックサム結合関数呼び出しおよびその他の関連オーバーヘッドを起動するのではなく、標準のコピーおよびチェックサム処理を使用する方が処理性能は高い。起動すべき時点等のための適切な公式を決定することは、各プロトコール毎になんらかの実験を必要とするであろうが、これは、ＳＴＲＥＡＭＳフレーム・ワークおよび本発明の実施形態とは無関係である。次の命令コードの例は、write()システム呼び出しと連係する本発明の動作に関するものである。

【００３６】

【表８】

```
/* もしも関数が登録されていれば、コントローラは適切な関数を実行する */
if (sth->sth_flags & F_STH_FUNC_REG_ENABLED) {
if (sth->sth_f_reg.sth_copyin) {
error = (*sth->sth_f_reg.sth_copyin)(mp, w_cnt, uiop,
sth->sth_copyin_threshold);
/* EAGAINであれば, copyinは実行されなかったので、通常のタスクを実行 */
if (error == EAGAIN)  UIOMOVE_WRITE(mp, cnt, uiop, error);
if (error)
goto error_processing_code)
}else{
UIOMOVE_WRITE(mp, cnt, uiop, error);
}
/* コントローラは呼び出し元が３２ビット・アプリケーションであって、
*かつ定義されていれば、３２から６４ビットへの変換を実行する。
*これはmblk上で実行される。
```

```
*/
if (sth->sth_f_reg.sth_32_to_64 && uarea->32bit_hint)
error = (*sth->sth_f_reg.sth_32_to_64)(mp);
/* その他の形式のオプション処理が出力方向データ上で実行されることを
*ドライバ／モジュールが望む場合、これをmblk上で実行する。
*/
if (sth->sth_f_reg.sth_write_opt)
error = (*sth->sth_f_reg.sth_write_opt)(mp);
}else{
/* 関数は定義されていないので、オリジナルのコードを実行する。
*/
}
```

【００３７】書込み経路コードの残りの部分は通常通り実行される。putmsg()/putpmsg()に関して、ＳＴＲＥＡＭＳフレーム・ワークは、次のように拡張される。putmsg()/putpmsg()は、M_PROTOまたはM__PCPROTOメッセージを用いて、より低位のモジュールまたはドライバに渡される制御データを呼び出し元が指定することを可能にする。このデータは、一旦それがカーネルに持ち込まれると、変換または後処理を必要とするかもしれない。このような場合の例は、ＴＰＩ(すなわちTransport Provider Interfaceトランスポート・プロバイダ・インターフェース)の３２ビット版を使用してコンパイルされた３２ビット・アプリケーションである。このインターフ＊ェースは、本発明によって３２ビットとしても６４ビットとしても正しく解読されるデータ・エレメントを含む構造を定義する。これらのデータ・エレメントは、６４ビット・システムが正しくデータ解釈を行うためには、６４ビット・データ要素に変換される必要がある。加えて、それがカーネルに一旦持ち込まれたなら、制御データはなにがしかの他の変換形式を必要とするかもしれない(注：write()経路上のような代替的copyin経路を経由して持ち込まれる通常のデータは変更されなくてよい)。適切な命令コードの１例は次の通りである。

【００３８】

【表９】

```
/* もし処理中であれば制御部分がカーネルに先ずコピーされる。
*通常のコピー(copyin)動作が実行され、次に後処理を次のように実行する。
*/
if (error = COPYIN(user_ptr, mp, cnt, ctl)){
/* エラーからの回復処理を実行する */
}
if (sth->sth_flags & F_STH_FUNC_REG_ENABLED) {
if (sth->sth_f_reg.sth_copyin) {
error = (*sth->sth_f_reg.sth_copyin)(mp, w_cnt, uiop,
sth->sth_copyin_threshold);
/* EAGAINであれば、copyinは実行されなかったので、通常タスクを実行する。
*/
if (error == EAGAIN)
error = COPYIN(user_ptr, mp, cnt, ctl);
if (error)
goto error_processing_code;
}else{
COPYIN(user_ptr, mp, cnt, ctl);}
/* 呼び出し元が３２ビット・アプリケーションであって、
*かつ定義されていれば、３２から６４ビットへの変換を実行する。
*/
if (sth->sth_f_reg.sth_32_to_64 && uarea->32bit_hint)
error = (*sth->sth_f_reg.sth_32_to_64)(mp);
/* その他の形式のオプション処理が出力方向データ上で実行されることを
*ドライバ／モジュールが望む場合、これをmblk上で実行する。
*/
```

```
if (sth->sth_f_reg.sth_write_opt)
error = (*sth->sth_f_reg.sth_write_opt)(mp);
}else{
/* 関数は定義されていないので、オリジナルのコードを実行する。
*/
```

【００３９】read()システム呼び出しに関しては、コードは本質的に同様であるが、その関数の機能性は、モジュール／ドライバ修正を必要とする関数機能性と置き換えられる。本発明は、３２ビット・ユーザ・プロセスの要求事項に従ってストリーム・データ解釈、ストリーム実行行動等に適合する関数を処理する６４ビット・カーネルのモジュールまたはドライバを備える。特に、この関数は、ユーザ・プロセスのスレッドを検査して３２ビット形式を必要としているのか６４ビット形式を必要としているのか判断する。ユーザ・プロセスが３２ビット形式を必要としているとすれば、該関数は、ユーザ・プロセスによって必要とされる３２ビット・データ形式と＊デバイス・ドライバによって必要とされる６４ビット・データ形式の間でメッセージのデータ形式を変換する。同様に、接続時に初期的に受け入れられたプロトコールの現在時使用バージョンと(例えばインターネット・プロトコル(ＩＰ)バージョン４ではなくインターネット・プロトコル(ＩＰ)バージョン６と)アプリケーションが対話できない場合は、データはその時点で正しいプロトコール形式に変換される。以下は、この関数のコードの１例である。

【００４０】

【表１０】

```
/* 関数が既に登録されていれば、コントローラは適切な関数を実行する。*/
if (sth->sth_flags & F_STH_FUNC_REG_ENABLED){
/* その他の形式のオプション処理が出力方向データ上で実行されることを
*ドライバ／モジュールが望む場合、これをmblk上で実行する。
*IPv4からIPv6への変換が実行されるのはこの場合の１例である。
*/
if (sth->sth_f_reg.sth_read_opt)
rror = (*sth->sth_f_reg.sth_read_opt)(mp);
}
/* 呼び出し元アプリケーションが３２ビット・アプリケーションで、
*その変換関すが登録されていれば、データをユーザ空間へ移動する前に
*その関数を起動する。
t application and there is a
*/
if (sth->sth_f_reg.sth_64_to_32 && uarea->32bit_hint)
error = (*sth->sth_f_reg.sth_64_to_32)(mp);
}
/* データ移動に関する現在時実施形態を使用してデータを移動する */
UIOMOVE_READ(mp, cnt, uiop, error);
```

【００４１】getmsg()およびgetpmsg() システム呼び出しに関連した本発明の使用は、上述のread()システム呼び出しの場合に類似しているが、メッセージの制御データ部分を取り扱うために追加のコードが含まれる。その※40

※コードの例は以下の通りである。

【００４２】

【表１１】

```
/* 関数が既に登録されていれば、コントローラは適切な関数を実行する。*/
if (sth->sth_flags & F_STH_FUNC_REG_ENABLED) {
/* その他の形式のオプション処理が出力方向データ上で実行されることを
*ドライバ／モジュールが望む場合、これをmblk上で実行する。
*IPv4からIPv6への変換が実行されるのはこの場合の１例である。
*/
if (sth->sth_f_reg.sth_read_opt)
error = (*sth->sth_f_reg.sth_read_opt)(mp);
/* 次に、メッセージのM_DAYTA部分に対して、３２ビット変換を、
*もしその関数が存在し登録されていれば、実行する。
```

```
*/
if (sth->sth_f_reg.sth_64_to_32 && uarea->32bit_hint)
error = (*sth->sth_f_reg.sth_64_to_32)(mp);
}
error = COPYOUT(mp, user_ptr, cnt, ctl);
```

【００４３】登録された関数処理が完了すると、データを通常の実行経路を使用するユーザ空間へ移動する。

【００４４】ioctl経路に関しては、データがカーネルから移動されているのかカーネルへ移動しているのかに従って、同様のステップが実行される。オプションとしてのioct関数がデータの移動を実行するものと、あるいは、データの移動を制限して上述の事前または事後処理の役割を単純化すると、実施上は仮定される。ioctlは異なるモジュールまたはドライバと通信することもできる点は理解されるべきであろう。従って、このオプション構造が利用されるならば、これらのメッセージがどのように解釈されるかという点に関する理解を持って(通常は最高位のモジュール／ドライバが最終アービタであるという理解の下で)、モジュールおよびドライバは構築される。このioctl経路が完全性の目的のために定義されているが、ioctlがメッセージ毎の関数を使用することが推奨される。なぜならば、ioctlを処理するモジュールまたはドライバの各々が特定のメッセージを目標とすることができるからである。コードの例は次の通りである。

【００４５】

【表１２】if (sth->sth_flags & F_STH_FUNC_REG_ENABLED &&sth->sth_f_reg.sth_ioctl_opt)
error = (*sth->sth_f_reg.sth_ioctl_opt)(mp, direction_hint);

【００４６】更に、本発明は関数を識別するモジュールまたはドライバを備え、カーネル内のモジュールまたはドライバの状態の動的変化に応答してストリーム・ヘッドにおける実行を制御する手段を提供する。例えば、本＊発明は、接続ネットワークから受け取る新しい情報を基にしたモジュールまたはドライバの優先度状態の変更に応答して、ストリーム・ヘッド上の待ち行列に既に保持されているメッセージの処理順位を通常から優先に変換する関数を識別する手段を提供する。これにより、要求される機能を提供するため、ユーザ・アプリケーションまたはインターフェース・ライブラリを修正する必要性がなくなる。

【００４７】本発明の代替実施形態において、ＳＴＲＥＡＭＳフレーム・ワークは、M_COPYOUTに関するread, getmsg, getpmsg, ioctlのような入力方向経路上でメッセージ毎に関数登録を実行するモジュールまたはドライバの能力をサポートするように拡張される。このような機能性が提供される理由は、モジュールまたはドライバが上流方向に向かうすべてのメッセージについて１つの関数セットが常に起動される必要はなく、ある番号Ｎのパケットだけについて必要とされるに過ぎない場合があるからである。このような機能性はシステムに柔軟性を持たせるために追加される。そのような関数を提供する根本的概念はなお同じである。そのような関数は、起動されたアプリケーションに基づいてデータを変換するために起動される。最も一般的用途は、本発明のパラメータについて常に例としてあげている６４ビット対３２ビットの変換のためである。第１に、ユーザ・データを記述するために使用されるメッセージ構造が適切に修正される。このデータ構造は<stream.h>内に定義される。

【００４８】

【表１３】

```
struct msgb {
struct msgb*      b-next;     /* 待ち行列上の次のメッセージ */
struct msgb*      b_prev;     /* 待ち行列上の前のメッセージ */
struct msgb*      b-cont;     /* 次のメッセージ・ブロック */
unsigned char*    b-rptr;     /* 最初の未読データ・バイト */
unsigned char*    b-wptr;     /* 再ション未書込データ・バイト */
struct datab*     b-datap;    /* データ・ブロック */
unsigned char     b-band;     /* メッセージ優先度 */
unsigned char     b-pad1;
unsigned short    b_flag;     /* メッセージ・フラグ */
(int32 *)         b_func;     /* 起動されるべき関数 */
MSG_KERNEL_FIELDS
};
```

【００４９】加えて、msgb->b_flagフィールド内に新しいフラグが定義される。この新しいフラグは、ユーザ・データがカーネルからユーザ空間へ次のアプリケーション・システム呼び出しを介して移動される時、msgb構造内に記憶されている関数がユーザ・データに対して起動されなければならないことを標示するために使用され

る。

【００５０】

【表１４】#define MSGFUNCREG 0x2000 /* データに組み込まれた関数を実行する。*/

【００５１】代替実施形態では、b_flagフィールドがセットされるが、どの登録メッセージ処理関数を起動すべきかを示すフラグが使用される。これはまたドライバ／モジュールが新しいmsgbフィールドb_device_id内にdevice_id値を含むことを必要とする。

【００５２】

```
/* メッセージ毎の処理がイネーブルされているか検査する。*/
if (mp->b_flag & MSGFUNCREG)
error = (*mp->b_func)(mp, uarea->32bithint);
/* 関数が登録されていれば、適切な関数を実行する。*/
if (sth->sth_flags & F_STH_FUNC_REG_ENABLED) {
...
}
```

【００５５】代替実施形態は次の通りである。

【００５６】

```
/* メッセージ毎の処理がイネーブルされているか検査する。*/
if (mp->b_flag & MSGFUNCREG){
error = (*(find_func(b_device_id, b_flag))(mp, uarea->32bithint);
/* 関数が登録されていれば、適切な関数を実行する。*/
if (sth->sth_flags & F_STH_FUNC_REG_ENABLED){
...
}
```

【００５７】これらのレジスタ関数ため種々の構造およびコーディングを使用することができる。以下は単純な構造の１例である。

```
int32
drv_a_32_to_64(MBLKP mp)
{
/* メッセージ・タイプを検査して実行すべき変換コードを決定する。
*メッセージ・タイプ毎に一組の変換コードが存在するであろう。
*各ドライバまたはモジュールが、アプリケーションとそれ自身の間で情報を
*通信するためにＡＢＩ(アプリケーション・バイナリ・インターフェース
*Application Binary Interface)を作成することが最適であろう。
*そうすることによって、すべてのメッセージ・タイプを理解するために厄介な
*コードを作成することなくメッセージ毎に変換を行うことができる。
*呼び出し元のアプリケーション・インスタンスが３２ビットで
*コンパイルされたアプリケーションであれば、この変換コードは
*入力方向６４から３２ビットへの経路上で変換を反転させるため
*再使用される。その他のタイプのオプション処理またはコピー／チェックサム
*に関しては、データの移動／操作がメッセージ毎のものであるので
*同様のコードが存在する可能性がある。例えば、ioctl経路について通常
*コピー・チェックサムを必要としない。代わって、所与のオプションとしての
*実行経路を実行する必要なしにユーザとカーネル間のデータ移動の間に
*データの自動変換を実行するcopyin／修正ユーティリティが必要と
*されるかもしれない。
*/
```

*【表１５】

```
#define MSGREADOPT    0x6000
#define MSGIOCTLOPT   0xa000
#define MSG64TO32     0xb000
```

【００５３】前述のコードは次のように修正される。read, getmsg, getpmsg, およびioctl経路に関して、コードは、コントローラによって実行される以下の２行を含む。

【００５４】

【表１６】

※【表１７】

★【００５８】

【表１８】

```
switch(mp->b_datap->db_type){
case M_DATA:
/* あらかじめ定義されたドライバＡＢＩに基づいて変換を実行 */
break;
case M_PROTO:
/* あらかじめ定義されたドライバＡＢＩに基づいて変換を実行 */
break;
case M_PCPROTO:
/* あらかじめ定義されたドライバＡＢＩに基づいて変換を実行 */
break;
case M_IOCTL:
/* あらかじめ定義されたドライバＡＢＩに基づいて変換を実行 */
break;
case M_COPYIN:
/* あらかじめ定義されたドライバＡＢＩに基づいて変換を実行 */
break;
case M_COPYOUT:
/* あらかじめ定義されたドライバＡＢＩに基づいて変換を実行 */
break;
default:
/* 何も実行しない */
}
}
```

【００５９】以上記述の通り、本発明は、メッセージ処理関数を識別するカーネル・レベルのモジュールまたはドライバを備え、ストリーム・ヘッドにおけるメッセージ処理関数の実行を制御する装置および方法を提供する。本発明の特定の実施形態を上記の通り記述したが、本発明はそのような実施形態の特定の形態または構成に限定されるべきではなく、本発明の有効範囲および精神を逸脱することなく上記実施形態に対する種々の修正および変更は可能である。

【００６０】本発明には、例として次のような実施様態が含まれる。

（１）デバイスとユーザ・プロセスの間でメッセージを通信するためのデータ経路を有するコンピュータ・システムにおいて、上記デバイスとメッセージを通信するため該デバイスに接続され、メッセージを処理するメッセージ処理関数を識別するように特に適合されたデバイス・ドライバと、上記ユーザ・プロセスとメッセージを通信するため上記デバイス・ドライバと上記ユーザ・プロセスの間に接続され、上記デバイス・ドライバによって識別された上記メッセージ処理関数の実行を制御する関数コントローラを含むように特に適合されたストリーム・ヘッドと、を備える装置。

（２）上記デバイス・ドライバが、上記メッセージ処理関数を識別する関数ポインタを備え、上記関数ポインタを上記ストリーム・ヘッドの上記関数コントローラに転送するように適合され、上記関数コントローラが上記関数ポインタを受け取るように適合された、上記（１）に記載の装置。

（３）上記ストリーム・ヘッドが、上記メッセージ処理関数の識別を記録するレジスタを含む、上記（１）に記載の装置。

（４）上記メッセージ処理関数が、メッセージのデータ形式を変換する関数を含む、上記（１）に記載の装置。

（５）上記メッセージ処理関数が、上記ユーザ・プロセスによって生成されたメッセージのデータ形式を決定するため上記ユーザ・プロセスのスレッドを検査する関数を含む、上記（１）に記載の装置。

【００６１】（６）上記メッセージ処理関数が、上記ユーザ・プロセスによって必要とされるい３２ビット・データ形式と上記デバイス・ドライバによって必要とされる６４ビット・データ形式の間でメッセージのデータ形式を変換する関数を含む、上記（１）に記載の装置。

（７）上記メッセージ処理関数が、メッセージのプロトコール形式を変換する関数を含む、上記（１）に記載の装置。

（８）上記メッセージ処理関数が、メッセージの各々に関してチェックサムを生成する関数を含む、上記（１）に記載の装置。

（９）上記ストリーム・ヘッドが、メッセージを保持する待ち行列を含み、上記メッセージ処理関数が、上記ストリーム・ヘッドの待ち行列上に保持されているメッセージを修正することによって上記デバイス・ドライバの状態の動的変化に応答する関数を含む、上記（１）に記載の装置。

（１０）上記ストリーム・ヘッドが、メッセージを保持するための待ち行列を含み、上記メッセージ処理関数が、上記ストリーム・ヘッドの上記待ち行列上に保持されているメッセージの優先度を修正することによって上記デバイス・ドライバの優先度状態の動的変化に応答する関数を含む、上記（１）に記載の装置。

（１１）コンピュータ・システムにおいて、メッセージ処理関数を識別するように特に適合されたストリーム処理モジュールと、ユーザ・プロセスとメッセージを通信するため上記ストリーム処理モジュールとユーザ・プロセスの間に接続され、ストリーム処理モジュールによって識別された上記メッセージ処理関数の実行を制御する関数コントローラを含むように特に適合されたストリーム・ヘッドと、を備える装置。

【００６２】（１２）コンピュータ・システムにおいてデバイスとユーザ・プロセスの間でメッセージを通信する方法であって、上記デバイスとデバイス・ドライバの間でのメッセージの通信と、上記デバイス・ドライバとストリーム・ヘッドの間でのメッセージの通信と、メッセージ処理関数の識別と、上記ストリーム・ヘッドにおける上記メッセージ処理関数の登録と、上記メッセージ処理関数の実行と、上記ストリーム・ヘッドと上記ユーザ・プロセスの間でのメッセージの通信と、を含むメッセージ通信方法。

（１３）上記メッセージ処理関数の識別が、メッセージ処理関数識別のためデバイス・ドライバの使用を含む、上記（１２）に記載のメッセージ通信方法。

（１４）上記デバイス・ドライバと上記ストリーム・ヘッド間のメッセージ通信が処理モジュールを経由して実行され、上記メッセージ処理関数の識別が、上記処理モジュールを使用して実行される、上記（１２）に記載のメッセージ通信方法。

（１５）上記メッセージ処理関数の実行が、メッセージのデータ形式変換を含む、上記（１２）に記載のメッセージ通信方法。

（１６）上記メッセージ処理関数の実行が、上記ユーザ・プロセスによって生成されたメッセージのデータ形式を決定するための上記ユーザ・プロセスのスレッドの検査を含む、上記（１２）に記載のメッセージ通信方法。

（１７）上記メッセージ処理関数の実行が、上記ユーザ・プロセスによって必要とされる３２ビット・データ形式と上記デバイス・ドライバによって必要とされる６４ビット・データ形式の間でのメッセージのデータ形式変換を含む、上記（１２）に記載のメッセージ通信方法。

（１８）上記メッセージ処理関数の実行が、メッセージのプロトコール形式変換を含む、上記（１２）に記載のメッセージ通信方法。

（１９）上記メッセージ処理関数の実行が、メッセージの各々に関するチェックサム生成を含む、上記（１２）に記載のメッセージ通信方法。

（２０）上記ストリーム・ヘッドの待ち行列におけるメッセージの保持を含み、上記メッセージ処理関数の実行が、上記ストリーム・ヘッドの待ち行列上に保持されているメッセージの修正による上記デバイス・ドライバの状態の動的変化への応答を含む、上記（１２）に記載のメッセージ通信方法。

【００６３】

【発明の効果】本発明は、ＳＴＲＥＡＭＳが実施されているシステム・アーキテクチャに従って従来固定的であったＳＴＲＥＡＭＳのシステム関数が、ユーザ・プロセスまたはデバイス・ドライバの動的要求に応答して適切な機能を提供することができる柔軟性を付与するという効果を奏する。これによって、例えば、６４ビット・カーネルおよびプロセッサ・アーキテクチャ上で３２ビット・カーネルおよびプロセッサ・アーキテクチャ上で実行される３２ビット・アプリケーションを、アプリケーションの変更を行うことなく、６４ビット・カーネルおよびプロセッサ・アーキテクチャ上で実行することが可能となる。

【図面の簡単な説明】

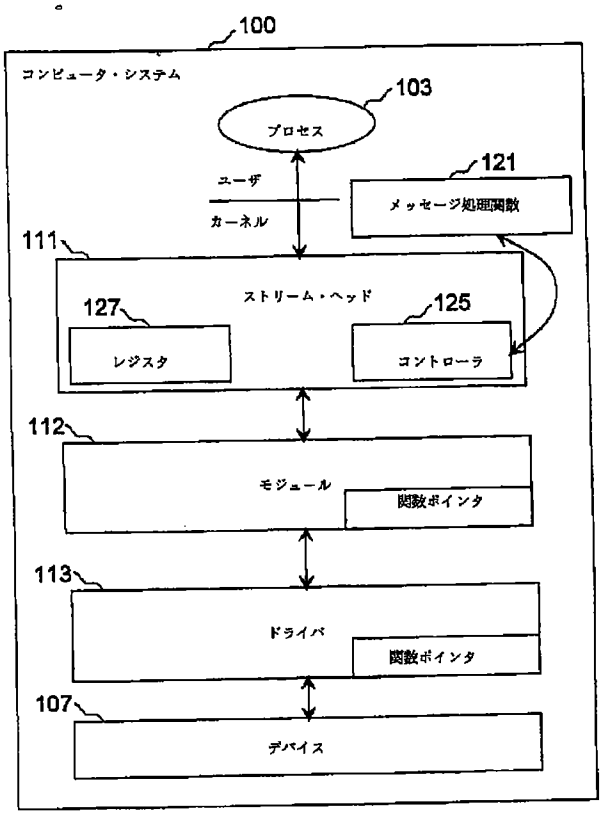
【図１】本発明の好ましい実施形態の１つを示すブロック図である。

【図２】本発明の好ましい実施形態の動作の流れを示す図である。

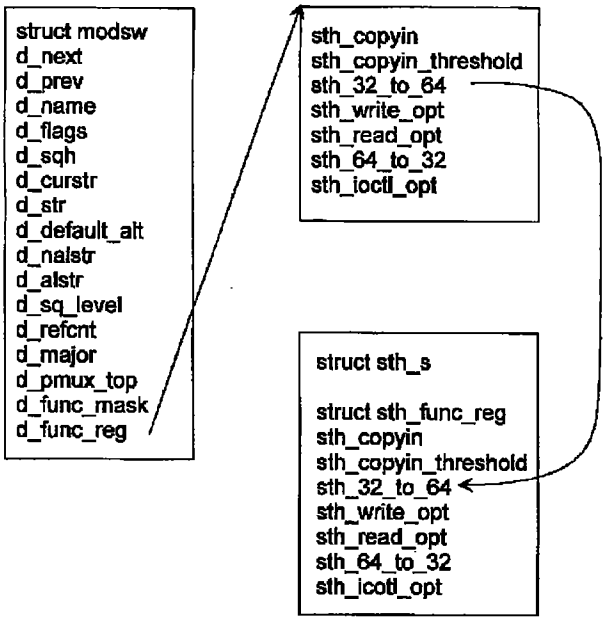
【符号の説明】

１００　コンピュータ・システム
１０３　ユーザ・プロセス
１０７　デバイス
１１１　ストリーム・ヘッド
１１２　モジュール
１１３　デバイス・ドライバ
１２１　メッセージ処理関数
１２５　コントローラ
１２７　レジスタ

【図1】


100
コンピュータ・システム
103
プロセス
121
ユーザ
カーネル
メッセージ処理関数
111
ストリーム・ヘッド
127
レジスタ
125
コントローラ
112
モジュール
関数ポインタ
113
ドライバ
関数ポインタ
107
デバイス

【図2】


struct modsw
d_next
d_prev
d_name
d_flags
d_sqh
d_curstr
d_str
d_default_alt
d_nalstr
d_alstr
d_sq_level
d_refcnt
d_major
d_pmux_top
d_func_mask
d_func_reg
sth_copyin
sth_copyin_threshold
sth_32_to_64
sth_write_opt
sth_read_opt
sth_64_to_32
sth_ioctl_opt
struct sth_s
struct sth_func_reg
sth_copyin
sth_copyin_threshold
sth_32_to_64
sth_write_opt
sth_read_opt
sth_64_to_32
sth_ioctl_opt

【公報種別】特許法第１７条の２の規定による補正の掲載
【部門区分】第６部門第３区分
【発行日】平成１３年１１月２２日（２００１．１１．２２）

【公開番号】特開平９－２２３０２７
【公開日】平成９年８月２６日（１９９７．８．２６）
【年通号数】公開特許公報９－２２３１
【出願番号】特願平８－３４０９０８
【国際特許分類第７版】
G06F 9/46 340
13/10 330
【ＦＩ】
G06F 9/46 340 A
C
13/10 330 B

【手続補正書】
【提出日】平成１３年４月１７日（２００１．４．１７）
【手続補正１】
【補正対象書類名】明細書
【補正対象項目名】特許請求の範囲
【補正方法】変更
【補正内容】
【特許請求の範囲】
【請求項１】　デバイスとユーザ・プロセスとの間でメッセージを通信するためのデータ経路を提供する、コンピュータシステム内に実現される装置であって、
前記メッセージの別のデータ表現を提供するデータ解釈関数であって、前記メッセージに対してデータ解釈演算を実行して前記メッセージの前記別のデータ表現を提供するようにされており、任意のＩ/Ｏリダイレクションデバイス・ドライバから独立に、そのようなデータ解釈演算を実際に実行する、データ解釈関数と、
前記デバイスとメッセージを通信するために該デバイスに接続され、前記データ解釈関数を識別するようにされているデバイス・ドライバと、
前記ユーザ・プロセスとメッセージを通信するために前記デバイス・ドライバと該ユーザ・プロセスの間に接続され、前記デバイス・ドライバによって識別される前記データ解釈関数のストリーム・ヘッドにおける実行を制御するためのデータ解釈関数コントローラを含むようにされているストリーム・ヘッドと、
を含む装置。
【請求項２】　メッセージの別のデータ表現を提供するためのデータ解釈関数と、
前記データ解釈関数を識別するようにされたストリーム処理モジュールと、
ユーザ・プロセスとメッセージを通信するために前記ストリーム処理モジュールと該ユーザ・プロセスの間に接続され、前記ストリーム処理モジュールにより識別される前記データ解釈関数の実行を制御するデータ解釈関数コントローラを含むようにされているストリーム・ヘッドと、
を含む、コンピュータシステム内に実現される装置。
【請求項３】　コンピュータシステム内部のデバイスとユーザ・プロセスの間でメッセージを通信する方法であって、
前記デバイスとデバイス・ドライバの間でメッセージを通信し、
前記デバイス・ドライバとストリーム・ヘッドの間でメッセージを通信し、
データ解釈関数を識別し、
前記ストリーム・ヘッドで前記データ解釈関数を登録して前記ストリーム・ヘッドにおける前記関数の実行を制御し、
前記データ解釈関数を実行することによって前記メッセージのデータ表現を変更し、
前記ストリーム・ヘッドにおける前記データ解釈関数の実行を制御し、
前記ストリーム・ヘッドと前記ユーザ・プロセスの間で前記メッセージを通信するようにした、方法。
【手続補正２】
【補正対象書類名】明細書
【補正対象項目名】０００９
【補正方法】変更
【補正内容】
【０００９】
【課題を解決するための手段】本発明は、カーネル・レベル・ストリーム処理モジュールまたはデバイス・ドライバに関するメッセージ処理関数（データ解釈関数）を

識別し、デバイスまたはユーザ・プロセスの要求のような種々のシステム要求に従ってストリーム・データ解釈、ストリーム実行行動等を適合するためそれら諸関数がストリーム・ヘッドにおいて使用可能となるように、ストリーム・ヘッドにおけるそれら識別された諸関数の実行を制御する装置および方法を提供する。